Weekly Work Report

Wenjie Niu

VISION@OUC

July 15, 2018

1 Content

1.1 Logistic Regression: Cost Function

To train the parameters ω and b, we need to define a cost function. In logistic regression, we want $\hat{y}^{(i)} \approx y^{(i)}$.

Loss(error) function

The loss function measures the discrepancy between the prediction($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function as shown in Eq. 1 2 computes the error for a single training example.

$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)}, y^{(i)})^2 \tag{1}$$

$$L(\hat{y}^{(i)}, y^{(i)}) = -[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \tag{2}$$

- If $y^{(i)} = 1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0

Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters ω and b that minimize the overall cost function as Eq. 3 while the cost function model in figure. 1.

$$J(\omega, b) = \frac{1}{m}\sigma[(y^{(i)}) \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})] \tag{3}$$

1.2 Gradient Descent

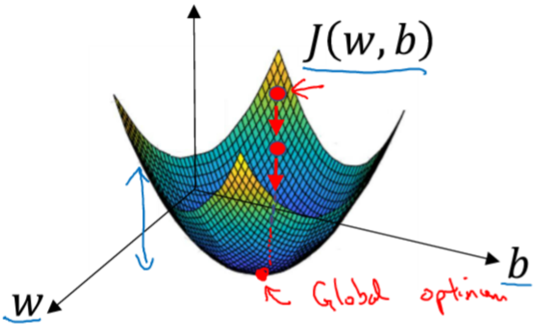


Figure 1: Gradient Descent Model

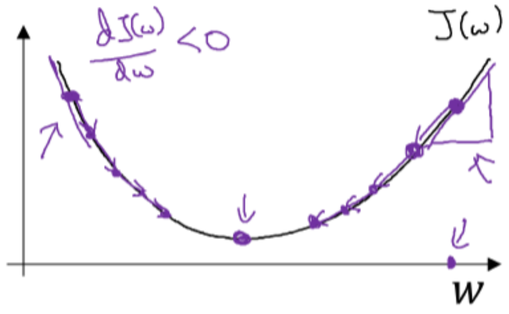


Figure 2: Gradient Descent Model in Two-dimension

If the gradient model is assumed in two-dimension(ω,$J(\omega)$) as the figure. 2 shown, it is convenient to observe the slope of the cost function at anyone point. In this case, the cost function in gradient descent is as Eq. 4

$$\omega := \omega - \alpha \frac{\partial J(\omega)}{\partial \omega} \tag{4}$$

Where " : " is equal to " update "; " α " represents learning rate; " ∂ " represents " derivative term ". Therefore, the cost function $J(\omega, b)$ in real case is as Eq. 56

$$\omega := \omega - \alpha \frac{\partial J(\omega, b)}{\partial \omega} \tag{5}$$

$$b := b - \alpha \frac{\partial J(\omega, b)}{\partial b} \tag{6}$$

1.3 Derivatives with a Computation Graph

The computations of a neural network are all organized in terms of a forward path or a forward propagation step in which we compute the output of the neural network followed by a backward pass or a back complication step which we use to compute gradients or computes derivatives. Take a function $J(a, b, c)$ as an example in Eq. 7.

$$J(a, b, c) = 3 \times (a + b \times c) \tag{7}$$

If $u = b \times c$, $v = a + v$, then finally the output $J = 3 \times v$ as shown in figure. 3. Sometimes, J is the cost function that we are trying to optimize. Through a left-to-right pause we can compute the value of J. When we are computing derivatives, it is a right-to-left pause. As we know before, in Calculus it actually called the chain rule which is the step of backward calculation. When you're writting codes to implement backpropagation, there usually be some fianl output variable that you really care about or you want to optimize. So in code we use in Eq. 8

$$dvar = \frac{\partial FindOutput}{\partial var} \tag{8}$$

to represent the derivative of the final output variable we care about such as J sometimes the last L with respect to the various intermediate quantities we're computing in code. This is same to the other variables such as $\frac{\partial J}{\partial u}=\frac{\partial J}{\partial v}\frac{\partial v}{\partial u}$, $\frac{\partial J}{\partial b}=\frac{\partial J}{\partial u}\frac{\partial u}{\partial b}$ and $\frac{\partial J}{\partial c}=\frac{\partial J}{\partial u}\frac{\partial u}{\partial c}$.

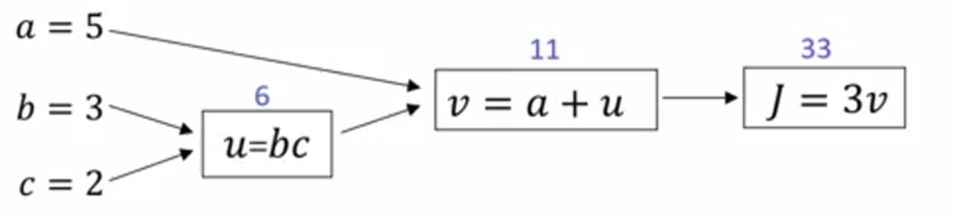


Figure 3: Gradient Graph

1.4 Logistic Regression Gradient Descent

This subsection is about how to implement gradient descent for logistic regression. Recap of logistic regression as follow Eq. 91011.

$$z = \omega^T x + b \tag{9}$$

$\hat{y}$ is defined as follows, where z is that and if we focus on just one example,then the loss or respect to that one example is defined as follows, where a is the output of the logistic regression and y is the ground truth label

$$\hat{y} = a = \sigma(z) \tag{10}$$

$$L(a, y) = -(y \log(a) + (1 - y) \log(1 - a)) \tag{11}$$

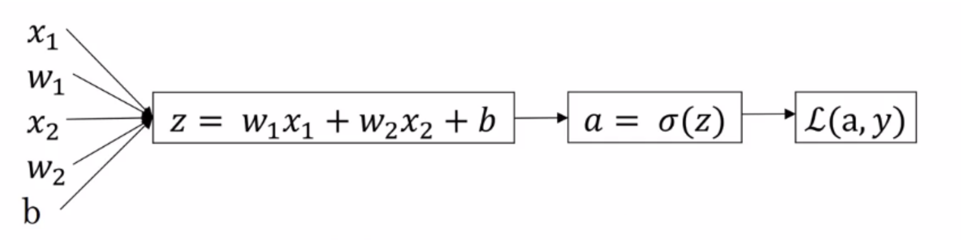


Figure 4: Gradient Graph

Another example as figure. 4, we have only two features x_1 and x_2, in order to compute z, we need to input ω_1, ω_2 and b, in addition to the feature values x_1 x_2. So these things in a computation graph get used to compute z. Finally we compute $L(a, y)$. In logistic regression what we want to do is to modify the parameters ω and b in order to reduce the loss.

If we want to compute derivatives represent to this loss, the first thing we want to do about going backwards is to compute the derivative of this loss with represent to this variable a while in code use da to denote this variable in Eq. 12.

$$da = \frac{\partial L(a, y)}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} \tag{12}$$

We can compute dz, $d\omega_1$, $d\omega_2$ and db using the same method as Eq: 13141516

$$d\omega_1 = \frac{\partial L}{\partial \omega_1} = a - y \tag{13}$$

$$d\omega_1 = \frac{\partial L}{\partial \omega_1} = x_1 \times dz \tag{14}$$

$$d\omega_2 = \frac{\partial L}{\partial \omega_2} = x_2 \times dz \tag{15}$$

$$db = dz \tag{16}$$

If we do it for m training examples, the definition of cost function $J(\omega, b)$ is this average

$$J(\omega, b) = \frac{1}{m} \sum_{i=1}^{m} L(a^{(i)}, y^{(i)}) \tag{17}$$

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(\omega^T x^{(i)} + b) \tag{18}$$

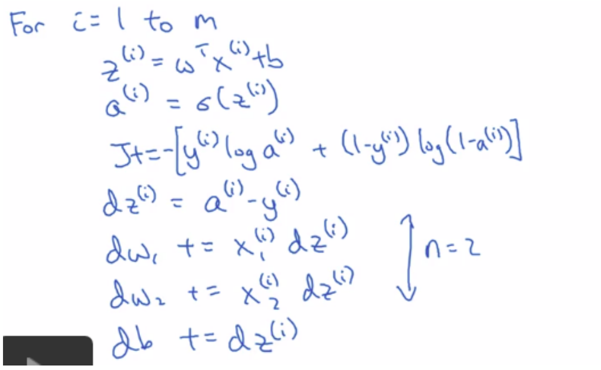
It turns out that the overall cost function with the sum was really the average of the 1 over m term of the individual losses like Eq. 1718. The derivative respect to say ω_1 of the overall cost function is also going to be the average of derivatives respect to ω_1 of the individual loss terms like 19

$$\frac{\partial J(\omega, b)}{\partial \omega_1} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial L(a^{(i)}, y^{(i)})}{\partial \omega_1} \tag{19}$$

Initialize J equals 0, $d\omega_1$ equals 0, ω_2 equals 0 and db equals 0. Use a for loop over the training set and compute the derivatives to respect each training example and then add them up all right like figure. 5. In this code, we use $d\omega_1$ and $d\omega_2$ as accumulators to sum over the entire training set whereas in contrust $dz_{(i)}$ is with respect to just one single training example, a superscript i refer to the one training example that's computed on. The result is in figure. 1.4.

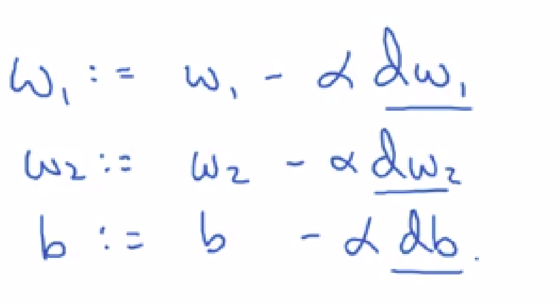
There are two weeknesses with the calculation as with implements.

- There are 2 for-loop:one is a small loop over the m training examples and the second for loop is over all n features.
- When implementing deep learning algorithms, having explicit for loop in code makes your algorithm run less efficiency.



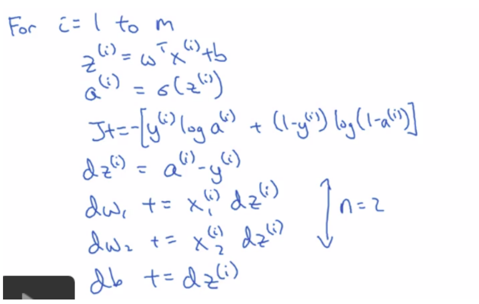
```
For $i = 1$ to $m$
    $z^{(i)} = w^T x^{(i)} + b$
    $a^{(i)} = \sigma(z^{(i)})$
    $J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$
    $dz^{(i)} = a^{(i)} - y^{(i)}$
    $dw_1 += x_1^{(i)} dz^{(i)}$    $\updownarrow n=2$
    $dw_2 += x_2^{(i)} dz^{(i)}$
    $db += dz^{(i)}$
```

Figure 5: Logistic Regression on m Examples

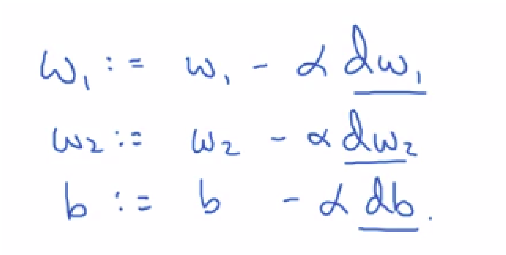


$$w_1 := w_1 - \alpha \underline{dw_1}$$
$$w_2 := w_2 - \alpha \underline{dw_2}$$
$$b := b - \alpha \underline{db}.$$

Figure 6: Logistic Regression on m Examples



```
For $i = 1$ to $m$
    $z^{(i)} = w^T x^{(i)} + b$
    $a^{(i)} = \sigma(z^{(i)})$
    $J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})]$
    $dz^{(i)} = a^{(i)} - y^{(i)}$
    $dw_1 += x_1^{(i)} dz^{(i)}$    $\updownarrow n=2$
    $dw_2 += x_2^{(i)} dz^{(i)}$
    $db += dz^{(i)}$
```



$$w_1 := w_1 - \alpha \underline{dw_1}$$
$$w_2 := w_2 - \alpha \underline{dw_2}$$
$$b := b - \alpha \underline{db}.$$

1.5 Vectorization

Vectorization is basically getting rid of explicit for loops in code. In practice of deep learning we need to train on relatively large data sets when deep learning algorithms tend to be shine. Therefore, the ability to perform vectorization have become a key skill. The non-vectorized implementation is in figure. 5. In contrast, a vectorized implementation in Python would just use the command as follow

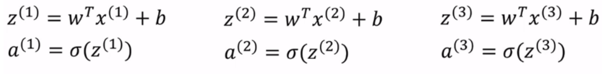
$$z = np.dot(\omega, x) + b \tag{20}$$

The approach 20 is much faster. It demonstrates that if you vectorize your code, it will be much faster. A lot of scaleable deep learning implementations are done on a GPU(Graphic Processing Unit). It turns out that both CPU and GPU have parallelizaiton instructions(sometimes called SIMD instructions) which stands for a single instruction multiple data.

If you use build-in functions such as $np.functions$ that don't require you exolicitly implementing a for loop. It enables Python numpy to take much better advantage of parallelism to do computations much faster. GPU is remarkably good at these SIMD calculations but CPU is actually also not too bad at that. The rule of thumb to remember is whenever possible, avoiding using explicit for-loops.

1.6 Vectorizing Logistic Regression

This part is about how we can vectorize the implementation of logistic regression, so they can process an entire training set that is implement a single iteration of gradient descent with respect to an entire training set without using even a single explicit for-loop. In order to carry out the forward propagation step as shown in figure. 7, it is necessary to compute these predictions on m training examples. There is a way to do so without needing an explicit for-loop.



$z^{(1)} = w^T x^{(1)} + b$ $\quad z^{(2)} = w^T x^{(2)} + b$ $\quad z^{(3)} = w^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)})$ $\quad a^{(2)} = \sigma(z^{(2)})$ $\quad a^{(3)} = \sigma(z^{(3)})$

Figure 7: The forward propagation steps of logistic regression

1.7 Vectorizing Logistic Regression's Gradient Computation

$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} \tag{21}$$

$$dw = \frac{1}{m} X dw^T \tag{22}$$

In Python Eq. 21 is $db = \frac{1}{m} np.sum(dz)$ and so on.

When implementing logistic regression, instead of looping over these, we can write them as

$Z = W^T + b$ in Python $Z = np.dot(W.T, x) + b$ $\quad A = \sigma(Z)$ $\quad dZ = A - y$ $\quad dw = \frac{1}{m} X s Z^T$ $db = \frac{1}{m} np.sum(dZ)$, these are backward propagation.

We should get rid of explicit for-loops, if you want to implement multiple as a gradient descent then you still need a for-loop over the number of iterations.

1.8 Broadcasting in Python

Broadcasting is another technique that make Python code run faster. If we had a 3 × 4 matrix and we divided it by a 1× 4 matrix. What Python will do is taking this number and auto-expand it in a 3× 4 vector then the matrix can be computed. In a word, there is a general principle in Python as shown in figure. 8

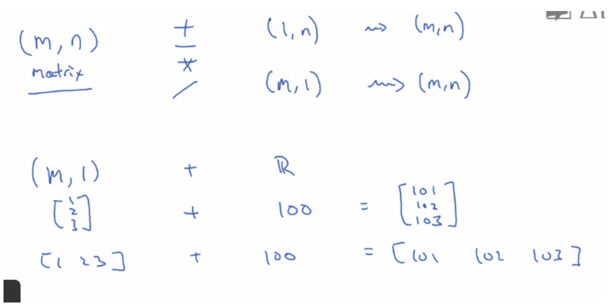
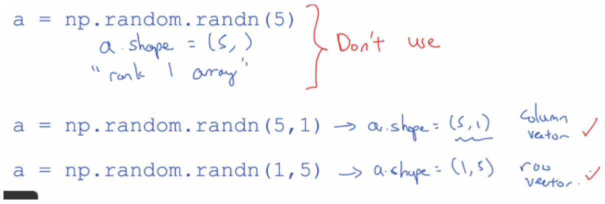


Figure 8: General Principle in Python

1.9 Python/numpy vector

$a = np.random.randn(5)$, its shape is $(5,)$ which is called a rank 1 array. It doesn't behave consistently as either a row vector nor a colimn vector. When doing programming exercises or implementing logistic regression on neural networks that we should not use these rank i arrays. Instead, when creating an array, we commit to making it either a column vector or a row vector as we can see in figure. 9. If not entirely sure what's the dimension of one of vectors, We could throw in an assertion statement like $a.assert(a, shape == (5, 1))$ to make sure that this is a vector in this case. And if for some reason you end up with a rank 1 array, you can reshape this like $a = a.reshape(1, 5)$ so that it behaves more consistently as either column vector or row vector.



```
a = np.random.randn(5)

a = np.random.randn(5,1)

a = np.random.randn(1,5)
```

Figure 9: Python/numpy vectors

2 Progress in this week

Step 1 Watched the courses clips.

Step 2 Wathced again and took notes.

Step 3 Grasped the related pictures and wrote the Latex.

Step 4 Organized the content and push to the github.

3 Plan

Objective: Learn Neural Network and Deep Learning by myself.

2018.07.15—2018.07.21 Watch week three course clips and take the note.

2018.07.22—2018.07.28 Do so on week four course.

2018.08.29—2018.08.04 Do so on next part course.

References

[1] A. Ng. Neural network and deep learning. http://mooc.study.163.com/smartSpec/detail/1001319001.htm.

[2] A. Ng. Neural network and deep learning. https://www.coursera.org/learn/neural-networks-deep-learning/exam/QR8kq/introduction-to-deep-learning.